

On Challenges and Opportunities in the Translation of Deep Neural Networks into Finite Automata

Marco Sälzer¹, Eric Alsmann¹ and Martin Lange¹

¹Faculty of Electrical Engineering and Computer Science, University of Kassel, Germany

Abstract

The certification of safety properties and interpretation of neural networks is a topic of great concern. We recently devised an automata-based approach for addressing these tasks. It translates a neural network with ReLU activations into a finite automaton, capturing the input-output relation induced by the neural network. We report on a proof-of-concept implementation of this translation for a subclass neural networks, so called binarized neural networks, pointing out difficulties and opportunities of the proposed approach.

Keywords

Neural Networks, Finite Automata, Verification and Interpretation

1. Introduction

Reliable methods for verification and interpretation tasks regarding properties of (Deep) Neural Networks (DNN) become increasingly important, due to the fact that DNN are used in a wide area of applications, including safety-critical ones. Recently, a unified framework for handling a broad range of such tasks was proposed in [1]. The idea is to translate a DNN N , computing $\mathbb{R}^m \rightarrow \mathbb{R}^n$, into a finite automaton, capturing the relation of pairs $(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^m \times \mathbb{R}^n$ induced by N in the form of multi-track words. Similarly, usual safety properties like adversarial robustness or output reachability [2] and interpretation properties like minimum sufficient reason [3] of DNN can be translated into problems on automata, allowing automata-theoretic tools to be used for the certification of such properties of DNN. We report on first experiments regarding the performance of the proposed translation from DNN to finite automata. In Sect. 2 and Sect. 3 we briefly introduce the theoretical foundations of this translation. In Sect. 4 we give an overview of our results. In Sect. 5, we discuss difficulties and opportunities of this framework.

2. Preliminaries


A DNN-node v is a computational unit, computing $v(\mathbf{x}) = \sigma(b_v + \sum_{i=1}^k w_i x_i)$ where $w_i \in \mathbb{R}$ are the *weights*, $b_v \in \mathbb{R}$ is the *bias*, $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ is an *activation function* and k is the *input dimension* of v . A DNN-layer l is a tuple $l = (v_1, \dots, v_l)$ of DNN-nodes with all v_i having the

OVERLAY 2023: 5th Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis, November 7, 2023, Rome, Italy

✉ marco.saelzer@uni-kassel.de (M. Sälzer); eric.alsmann@uni-kassel.de (E. Alsmann); martin.lange@uni-kassel.de (M. Lange)



© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

same input dimension. A DNN N is a sequence of layers l_1, \dots, l_L such that for all l_i, l_{i+1} the input dimension of nodes in l_{i+1} is equal to the size of l_i . The function computed by N is given by $l_L(l_{L-1}(\dots l_1(\mathbf{x}) \dots))$. In this work, we only consider DNN in which the activation function at any node is the *ReLU function* $\text{relu}(\mathbf{x}) = \max(0, \mathbf{x})$.

Let Σ be a finite alphabet. We denote the *multi-track alphabet* consisting of k -vectors of symbols from Σ by Σ^k . A *multi-track (nondeterministic) finite automata (NFA)* is defined as a usual NFA with transitions over some Σ^k . Similarly, the language of such a multi-track NFA is defined as usual. In the remainder of this paper the term NFA refers to such multi-track NFA over a number of tracks k , which is clear from the context.

3. From Deep Neural Networks to NFA over Multi-Track Words

Let N be some DNN computing $\mathbb{R}^m \rightarrow \mathbb{R}^n$, defining the relation $R_N = \{(\mathbf{x}, \mathbf{y}) \mid \mathbf{x} \in \mathbb{R}^m, \mathbf{y} = N(\mathbf{x})\}$ of all its input-output pairs. Now, [1] presents a construction of a finite automaton A_N over a multi-track alphabet Σ^{m+n} such that the language $L(A_N)$ of A_N captures R_N in the sense that each word in $L(A_N)$ can be decoded into a pair $(\mathbf{x}, \mathbf{y}) \in R_N$ and each such pair of R_N can be encoded into a word $w \in L(A_N)$. An obvious way to encode a tuple (\mathbf{x}, \mathbf{y}) as a word over some Σ^{m+n} is based on the usual binary encoding, leading to the use of automata over infinite words. It can be observed that weak nondeterministic Büchi automata suffice [4, 1]. Here in this work we focus on finitely representable values: we assume that N computes a function over $(\mathbb{Z} \setminus 2^i)^m \rightarrow (\mathbb{Z} \setminus 2^i)^n$ where $(\mathbb{Z} \setminus 2^i)$ are the *dyadic rational numbers*, which are exactly those that have a finite binary representation. Furthermore, we assume that all weights and biases used in N are also from $(\mathbb{Z} \setminus 2^i)$.¹ Then, we can represent each (\mathbf{x}, \mathbf{y}) using a finite word over Σ^{m+n} with $\Sigma = \{+, -, 0, 1, .\}$, used henceforth for the remainder of this article. For example, the vector $(-3.725, 9.125)$ is represented by

$$w = \begin{bmatrix} - \\ + \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} . \\ . \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

We denote the function mapping such a multi-track word to the unambiguous vector of values from $(\mathbb{Z} \setminus 2^i)$ by *dec*. Then, one of the core results of [1] transfers easily.

Theorem 1. *Let N be a DNN with input dimension m and output dimension n . There is NFA A_N s.t. $L(A_N) = \{w \in (\Sigma^{m+n})^* \mid N(\text{dec}(w_1), \dots, \text{dec}(w_m)) = (\text{dec}(w_{m+1}), \dots, \text{dec}(w_{m+n}))\}$.*

Proof sketch. First observe that a DNN-node computes its output using four basic operations: 1. addition of two values; 2. multiplication of a value with a fixed constant; 3. addition of a value with a constant; and 4. the application of the ReLU function. The relation of value triples and tuples induced by each of these operations can be recognized by an NFA. We refer to [1] for details with the indication that the constructions are similar for our setting. An example of an NFA recognising the addition relation is given in Figure 1. Second, observe that we can combine these NFA using join and projection operations on single tracks to build an NFA recognising the

¹This restriction is no limitation considering the presumed applications of this translation, namely the verification and interpretation of DNN used in practice. There DNN necessarily contain and work over finitely representable parameters and values.

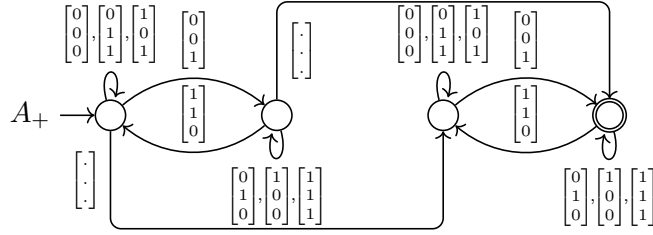


Figure 1: Sketch of an NFA that recognises the addition relation. For brevity reasons, we left out parts of the automaton handling different sign symbols of words from $(\Sigma^3)^*$ and parts ensuring that each word has at least a single bit after the dot symbols.

input-output relation defined by a DNN-node. Then, we use the same operations to establish an inductive procedure to generate an NFA recognising the input-output relation defined by a DNN. This is possible due to the inductive structure of DNN as indicated in Sect. 2. Again, see [1] for details. \square

4. A Proof-of-Concept Implementation

We present a proof-of-concept implementation, translating so called *Binarized Neural Networks* (BNN), into input-output equivalent NFA as discussed in Sect. 3. The BNN model [5] evolved as a simple but powerful neural network model, defined similar to DNN but with weights and bias restricted to $+1$ and -1 . It was shown, that BNN nearly achieve state-of-the-art performances in comparison to standard DNN [6].

The code is written in Python.² The construction of an NFA A_N for a given BNN conceptually follows the inductive construction described in Sect. 3 but uses two optimisations: first, we represent transition labels symbolically. Consider the NFA for addition in Fig. 1. It is notable that transitions are invariant under permutations of the labels in tracks one and two. This is not surprising as addition is commutative. Consequently, the transition labels in these automata can be stored symbolically as a pair (r, n) with $r \in \{0, 1\}$ representing the resulting bit in the addition, and $n \in \mathbb{N}$ representing the number of 1-bits in the tracks to be added up. Second, we use minimisation. After each construction that may create a non-minimal automaton, explicit minimisation is employed. Since NFA minimisation is PSPACE-hard in general, we use bisimulation quotienting instead. It may not produce minimal automata but it can be done in low polynomial time [7], and it typically yields good results in practice [8].

As benchmarks we use semi-randomly generated BNN: the input size is 1, the output size of the BNN is 1 and all layers in between consist of 2 nodes. We denote such BNN with BNN_{fix} . The parameters (weights and bias) of the considered BNN_{fix} were chosen uniformly from $\{-1, 1\}$. The benchmarks were computed on an Apple M1 Pro CPU with 32GB RAM. In the first part, we run our translation without intermediate minimisation steps for BNN_{fix} as specified above with 1 to 5 layers. The results are presented in the left chart of Fig. 2. The number of layers is plotted

²Available via <https://github.com/marcosaelzer/NN2NFA>

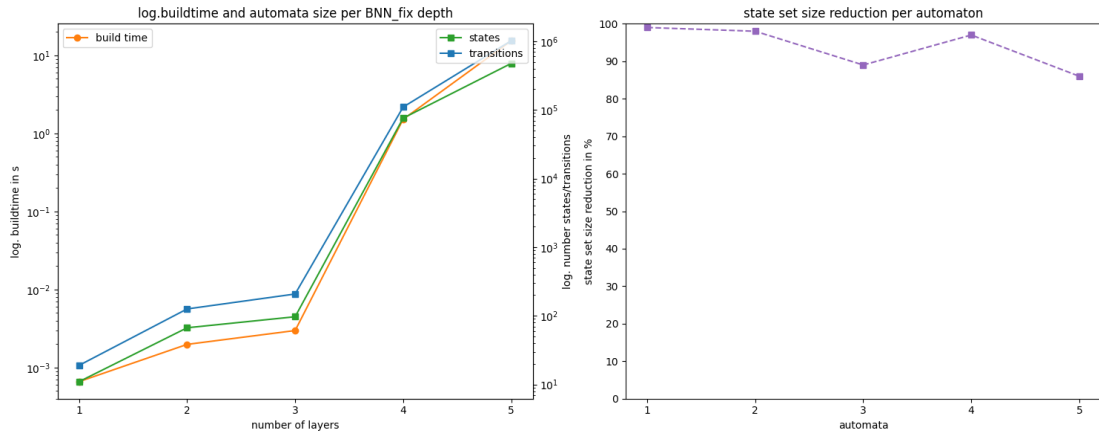


Figure 2: Left chart: logarithmic scaled buildtime, number of nodes and number of transitions of the NFA resulting from translating randomly generated BNN_{fix} with increasing number of layers, from 1 to 5. Right chart: percentage of state set size reduction due to minimisations on 5 randomly generated BNN_{fix} each with 4 layers.

on the horizontal axis and the runtime, number of states/transitions on the two logarithmically scaled vertical axes. The exponential blowup is clearly indicated by the linear interpolations of the three data series, even for these relatively shallow BNN_{fix} . The fluctuations are due to the varying amount of -1 and 1 parameters. In the right chart of Fig. 2 we see the effect of minimisation. We considered 5 randomly generated BNN_{fix} with 4 layers. The chart depicts the percentage of reduction of the state set due to minimisation. The state size reduction is $>80\%$, which indicates that our construction leads to an enormous amount of redundant states.

5. Discussion and Outlook

We presented a proof-of-concept implementation of the translation from DNN into finite automata, here NFA over multi-track words, as introduced in [1]. The translation is exponential, necessarily so since it reduces NP-hard DNN-reachability [9] to the NLOGSPACE problem for emptiness of (weak) NBA. Unsurprisingly, this blow-up is visible in the benchmarks. In order to achieve manageable runtimes, we focused our implementation on BNN of small size. There are two takeaways from this restriction: first, the maximum representation size of a parameter in the original DNN is crucial for the performance of the translation and, second, the inductive approach presented in [1] leads to many redundant states as indicated by the high amount of reduction achieved by minimising the NFA in intermediate steps. This opens a clear path for future work. The presumed applications of the DNN to NFA translation, namely verification and interpretation of DNN, are clearly tailored for DNN with parameters of small size, for example Quantized Neural Networks [10]. Furthermore, the inductive translation approach seems intractable, due to the amount of redundant states generated and, thus, future work must focus on establishing a translation of a more direct manner.

References

- [1] M. Sälzer, E. Alsmann, F. Bruse, M. Lange, Verifying and interpreting neural networks using finite automata, 2022. [arXiv: 2211.01022](https://arxiv.org/abs/2211.01022).
- [2] X. Huang, D. Kroening, W. Ruan, J. Sharp, Y. Sun, E. Thamo, M. Wu, X. Yi, A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability, *Comput. Sci. Rev.* 37 (2020) 100270. doi:10.1016/j.cosrev.2020.100270.
- [3] P. Barceló, M. Monet, J. Pérez, B. Subercaseaux, Model interpretability through the lens of computational complexity, in: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*, December 6-12, 2020, virtual, 2020. URL: <https://proceedings.neurips.cc/paper/2020/hash/b1adda14824f50ef24ff1c05bb66faf3-Abstract.html>.
- [4] B. Boigelot, S. Rassart, P. Wolper, On the expressiveness of real and integer arithmetic automata (extended abstract), in: *Automata, Languages and Programming, 25th International Colloquium, ICALP'98*, Aalborg, Denmark, July 13-17, 1998, Proceedings, volume 1443 of *Lecture Notes in Computer Science*, Springer, 1998, pp. 152–163. doi:10.1007/BFb0055049.
- [5] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, Y. Bengio, Binarized neural networks, in: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016*, December 5-10, 2016, Barcelona, Spain, 2016, pp. 4107–4115. URL: <https://proceedings.neurips.cc/paper/2016/hash/d8330f857a17c53d217014ee776bfd50-Abstract.html>.
- [6] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, Y. Bengio, Quantized neural networks: Training neural networks with low precision weights and activations, *Journal of Machine Learning Research* 18 (2018) 1–30. URL: <http://jmlr.org/papers/v18/16-456.html>.
- [7] R. Paige, R. Tarjan, Three partition refinement algorithms, *SIAM Journal on Computing* 16 (1987) 973–989.
- [8] J. Högberg, A. Maletti, J. May, Backward and forward bisimulation minimization of tree automata, *Theoretical Computer Science* 410 (2009) 3539–3552. doi:<https://doi.org/10.1016/j.tcs.2009.03.022>, implementation and Application of Automata (CIAA 2007).
- [9] M. Sälzer, M. Lange, Reachability is NP-complete even for the simplest neural networks, in: *Proc. 15th Int. Conf. on Reachability Problems, RP'21*, volume 13035 of *LNCS*, Springer, 2021, pp. 149–164. doi:10.1007/978-3-030-89716-1_10.
- [10] T. Liang, J. Glossner, L. Wang, S. Shi, X. Zhang, Pruning and quantization for deep neural network acceleration: A survey, *Neurocomputing* 461 (2021) 370–403. URL: <https://doi.org/10.1016/j.neucom.2021.07.045>. doi:10.1016/j.neucom.2021.07.045.